World Coordinate System and Transformation Pipeline

Project 3D Graphics Engine Design

CMPE 240

Name: Harish Marepalli

SJSU ID: 016707314

Email: harish.marepalli@sjsu.edu

Team Name: Student Group_1

Team Members: Tirumala Saiteja Goruganthu, Debasish Panigrahi, Shahnawaz Idariya

Target Board: LPC1769

INTRODUCTION:

This project's objective is to create and display the world coordinate system and two 3D objects (one is a cube, and the other is a half sphere) based on the transformation pipeline. This project is a part of the 3D graphics engine design on ARM Cortex CPU with C program emulation on LPC1769 using an SPI LCD display device.

This program is designed in such a way that the LPC1769 controls the pixel content displayed on the LCD. The program is implemented in C programming language which displays a 3D cube and a half sphere. Successful validation of the 3D graphics processing engine design is shown on the ST7735 LCD module.

This report consists of block diagrams of the components used and screenshots of the output. The LPC module is connected to the computer's USB port to get the required power supply. The LCD module draws in 3.3v from the LPC built in power supply. In LPC1769, pin numbers 11,12,13,14 are used for communication as we are using SPI port number 0 for our project code. The LCD peripheral is connected via appropriate pins with the LPC module's SPI port for communication.

1. System block diagram of the setup

The figure 1 shown below is the top-level system block diagram or system layout including laptop computer. The power is supplied from the computer to LPC module via a USB cable. The LCD display module is powered from the VCC and GND pins from the LPC1769 module.

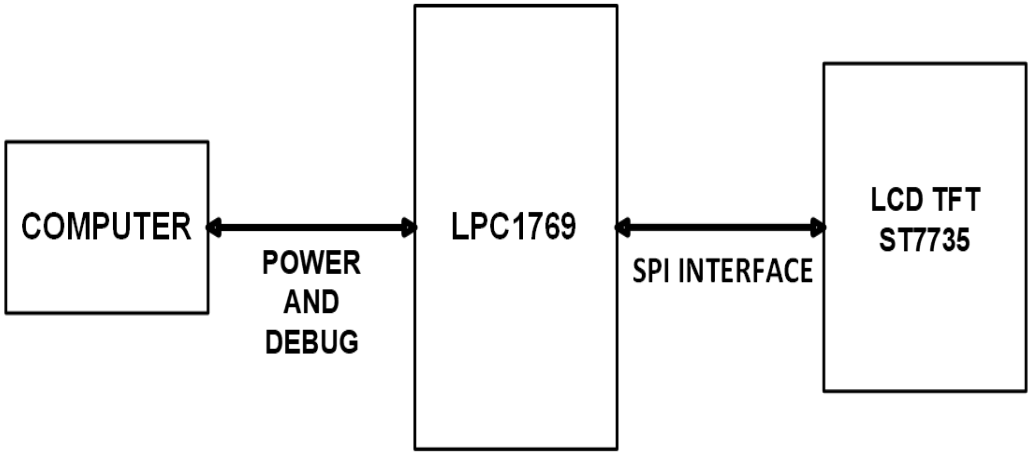

Fig. 1: System block diagram of the setup

1a. Items Used

i. CPU Module
ii. LCD Display
iii. PC

2. System block diagram of the SPI color LCD interface

The Figure 2 shown below is the system block diagram of the SPI color LCD interface. The SPI interface circuit is inbuilt on the wire wrapping board of a certain size. It consists of an external LCD and LPC1769 module. The circuit is connected to the computer using the USB cable.

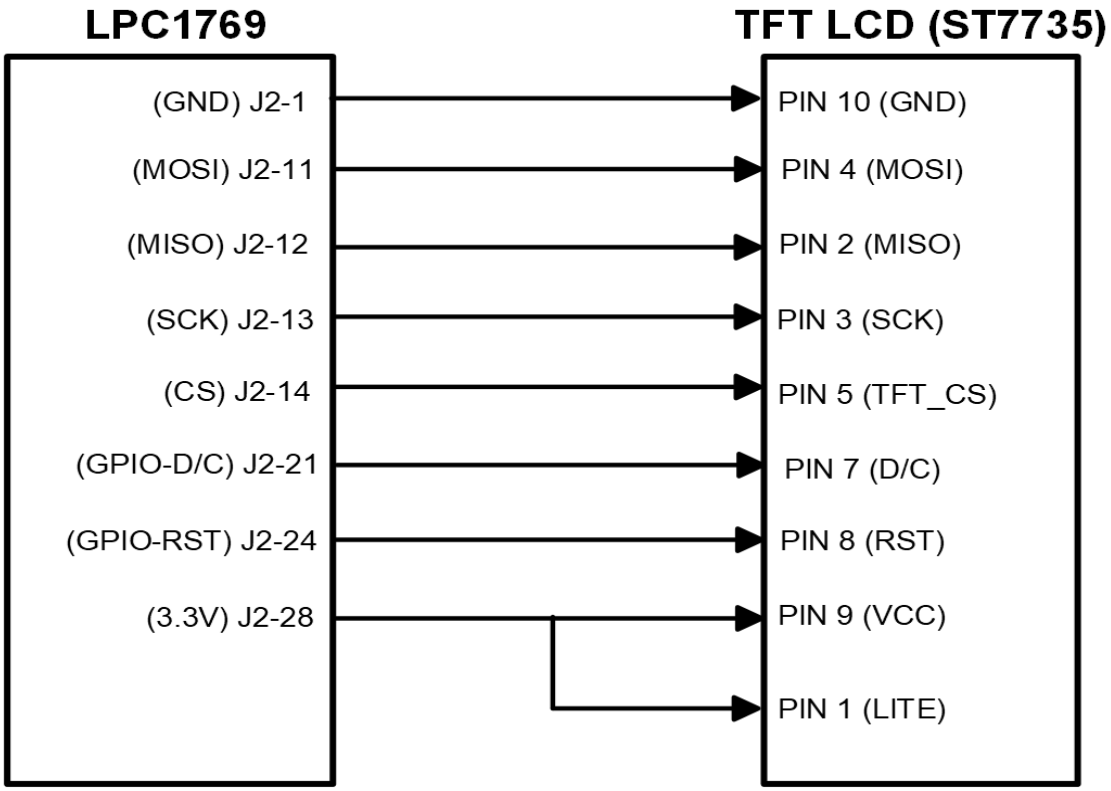

Fig. 2: System block diagram of the SPI color LCD interface

3. Schematics of the LPC1769 interface to LCD color display panel

The figure 3 below is the schematics of the LPC1769 interfacing with LCD color display panel. LPCXpresso module is connected to the 1.8" 18-bit color TFT LCD display to ensure the functionality working as expected. This module is used to display images and shapes of geometric figures. It uses 4-wire SPI to communicate and has its own pixel addressable frame buffer. The resolution of the LCD display is 128 by 160. It includes a 3.3V regulator for low voltage drop out and a 3/5 level shifter circuit so that we can have input power logic of 3V or 5V. The LPC module and LCD display are built on the wire wrapping board and are connected to the required pin via connecting wires. Once the required hardware is built, MCUXpresso IDE is utilized to program the LPC module.

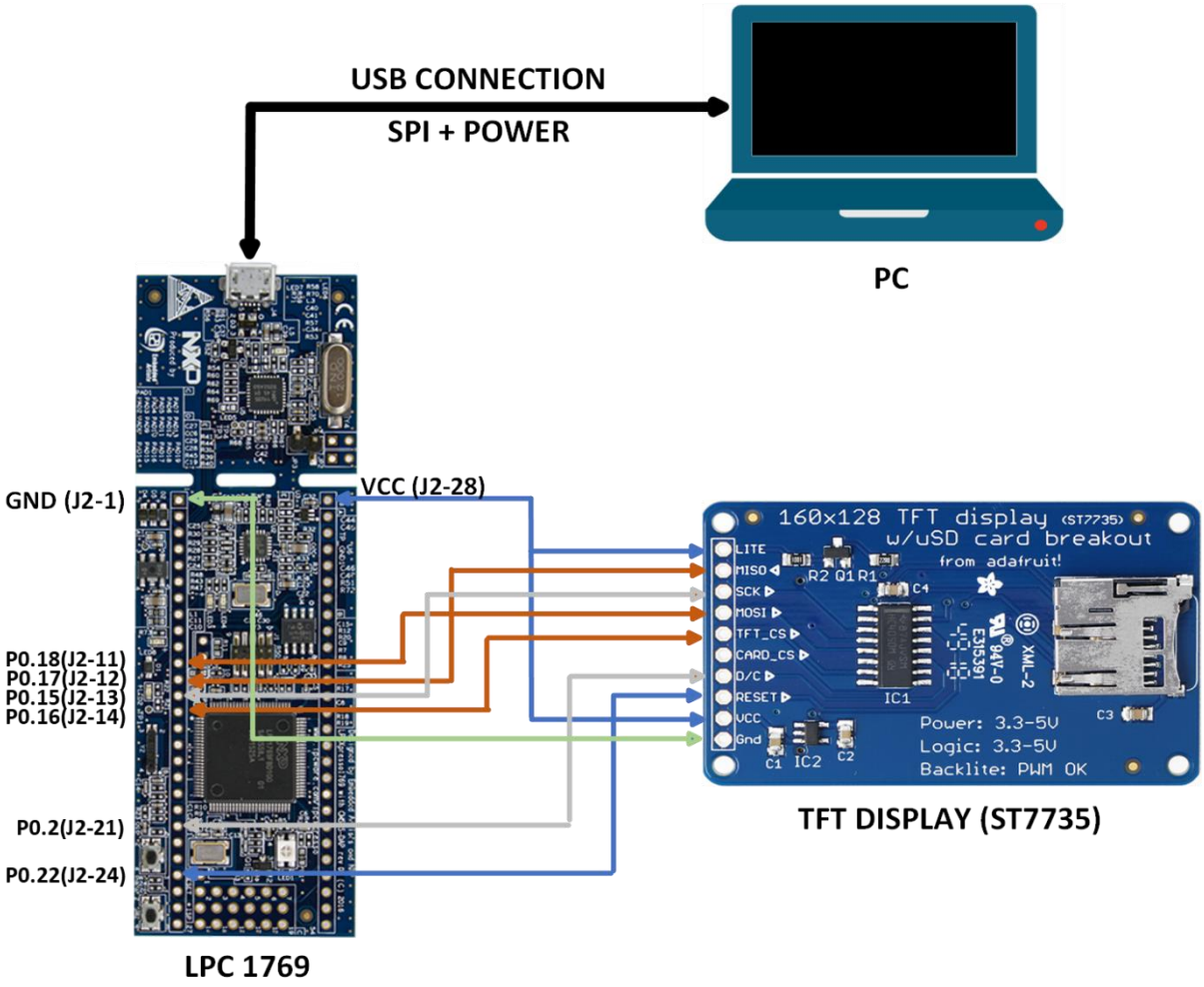

Fig. 3: Schematics of the LPC1769 interface to LCD color display panel

3a. LPC1769 PINOUT

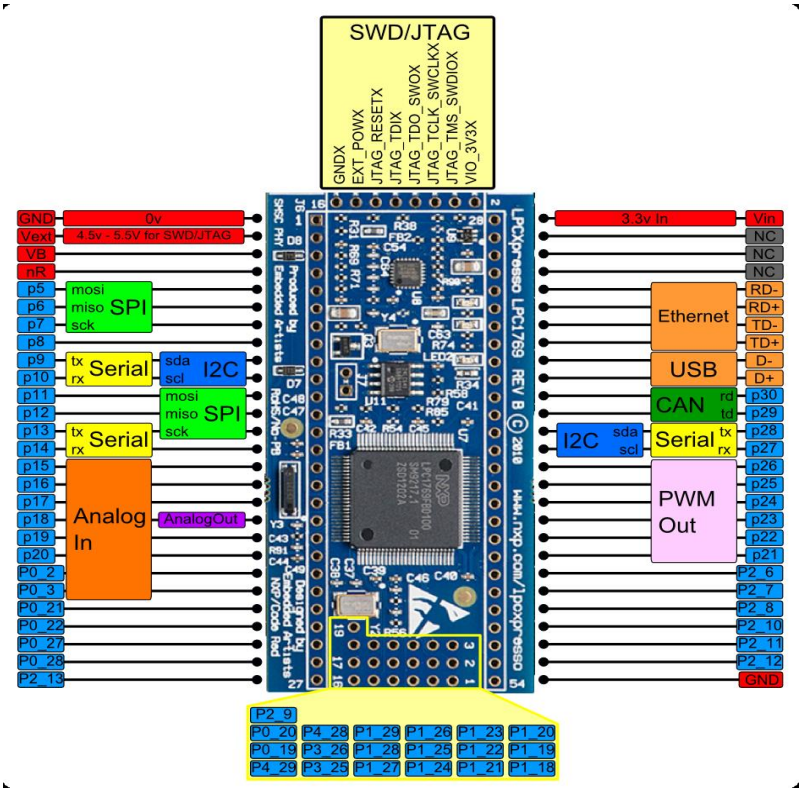

Fig. 3a: LPC1769 PINOUT

Figure 3a shows the LPC1769 pinout diagram.

3b. LCD display and its pinout

Figure 3b shows the LCD display and Figure 3c shows its pinout. The ST7735 is a single chip controller/driver for 262K-color, graphic type TFT-LCD. It consists of 396 source line and 162 gate line driving circuits. This chip is capable of connecting directly to an external microprocessor, and accepts Serial Peripheral Interface (SPI) 8, 12, 16, and 18-bit parallel interface. Display data can be stored in the on-chip display data RAM of 132 x 162 x 18 bits. It can perform display data RAM read/write operation with no external operation clock to minimize power consumption.

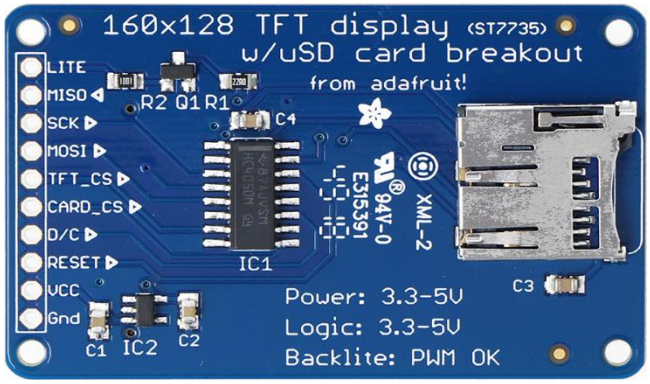

Fig. 3b: 160x128 TFT Display with ST7735

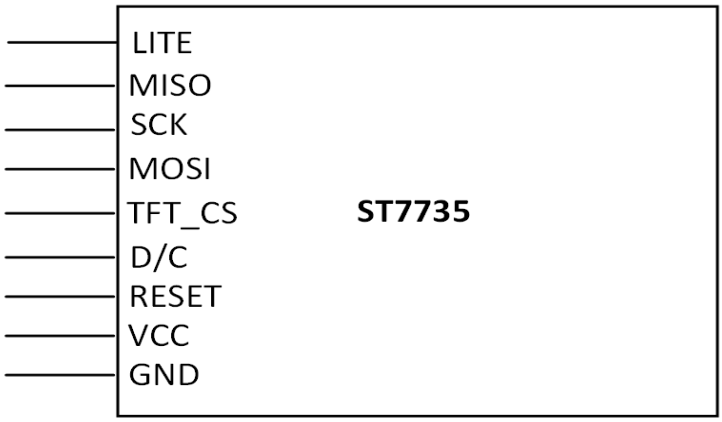

Fig. 3c: LCD Pinout

Some of the technical specifications for the Liquid Crystal Display Module are:

- 1.8” diagonal LCD TFT display.
- 128x160 resolution.
- 18-bit (262,144) color.
- 4 or 5 wire SPI digital interface.
- 2 white LED backlight with PWM dimmer.
- 3.3V/5V compatible power logic.
- Overall dimensions: 34mm x 56mm x 6.5mm.
- Current drawn: 50mA (full backlight)
- Manufactured by Adafruit.
- ST7735 controlled with built in pixel-addressable RAM buffer video.

4. Pin connectivity table

We setup the connections between the LCD and LPC module. The SPI interface circuit is inbuilt on the wire wrapping board of a certain size. It consists of an external LCD and an LPC1769 module. The circuit is connected to the computer using the USB cable. The connections are shown in the table 1 below.

CPU MODULE	TFT LCD DISPLAY
+3V3/J2-28	LITE
MISO0/P0.17/J2-12	MISO
SCK0/P0.15/J2-13	SCK
MOSI0/P0.18/J2-11	MOSI
SSEL0/P0.16/J2-14	TFT_CS
NA	CARD_CS
P0.2/J2-21	D/C
P0.22/J2-24	RESET
+3V3/J2-28	VCC
GND/J2-1	GND

Table 1: Pin Connectivity between LPC and LCD

Additional Points:

Software Used:

1. Coding is done in which data is sent and processed by LCD in the form of frames.
2. Every frame consists of a start of a frame and end of the frame for the LCD to differentiate the incoming data.
3. Usually, the sequence would start with a header byte followed by the command and then the data.
4. At the end, a frame tail would be present.
5. On the whole, the data in each frame is limited to 249 bytes maximum as the total including the header, command, and tail will come up to 255 bytes.
6. The program is built using Adafruit graphics library and MCUXpresso IDE.
7. The software requirements involve initializing the LCD module using Adafruit library for sending data buffer to the LCD as well as polling the GPIO pins for any external interrupt and based on the external interrupt taking actions for displaying content on the LCD screen.
8. We need the following to fulfil the software requirement for successful implementation:
    a. Windows
    b. MCUXpresso IDE Version 11
    c. External LCD Communication Programming Opcode.

Formulae:

World-to-viewer transform:

$$\mathbf{T} = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\cos\phi\cos\theta & -\cos\phi\sin\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\cos\theta & -\cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can write equations from this matrix.

1. $x_i' = -\sin\theta * x_i + \cos\theta * y_i$
2. $y_i' = -\cos\phi * \cos\theta * x_i - \cos\phi * \sin\theta * y_i + \sin\phi * z_i$
3. $z_i' = -\sin\phi * \cos\theta * x_i - \sin\phi * \cos\theta * y_i - \cos\phi * z_i + \rho$

Perspective Projection:

$$x_p = x_e\left(\frac{D}{z_e}\right)$$

$$y_p = y_e\left(\frac{D}{z_e}\right)$$

5. Screen Captures

Below are the images of implementation.

1. Figure 4 shows the Prototype board front side. The prototype board is used to connect LPC1769 with the LCD display.

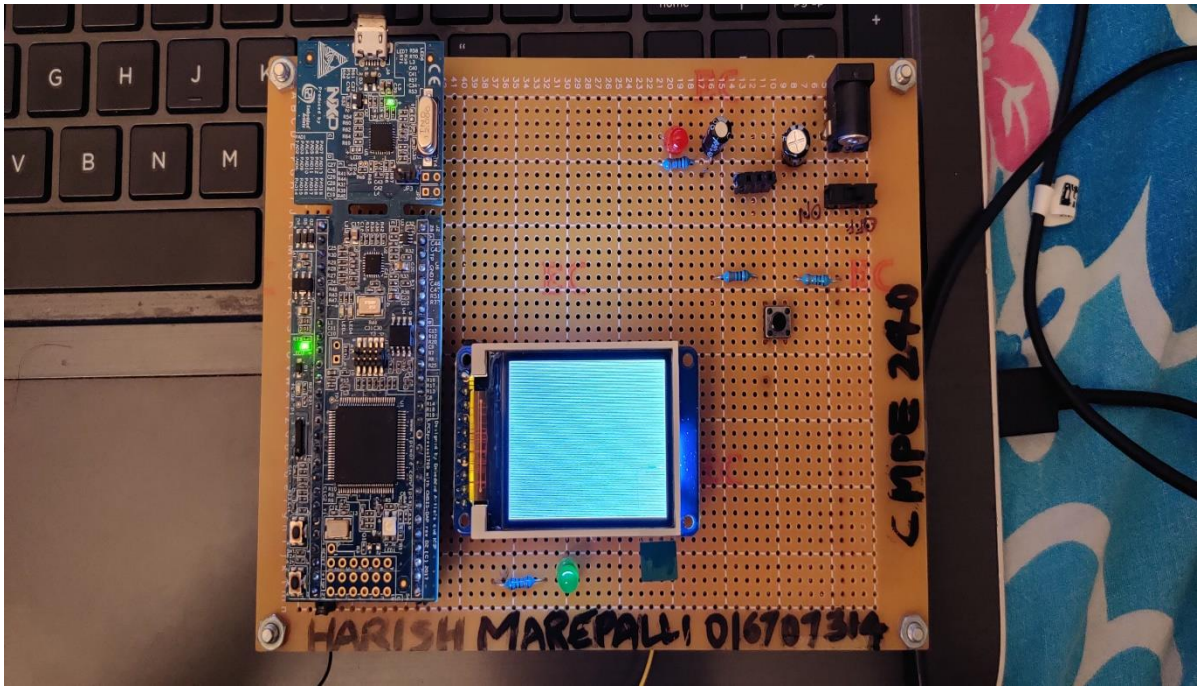

Fig. 4: Prototype Board Front

2. Figure 5 shows the Prototype board back side connections.

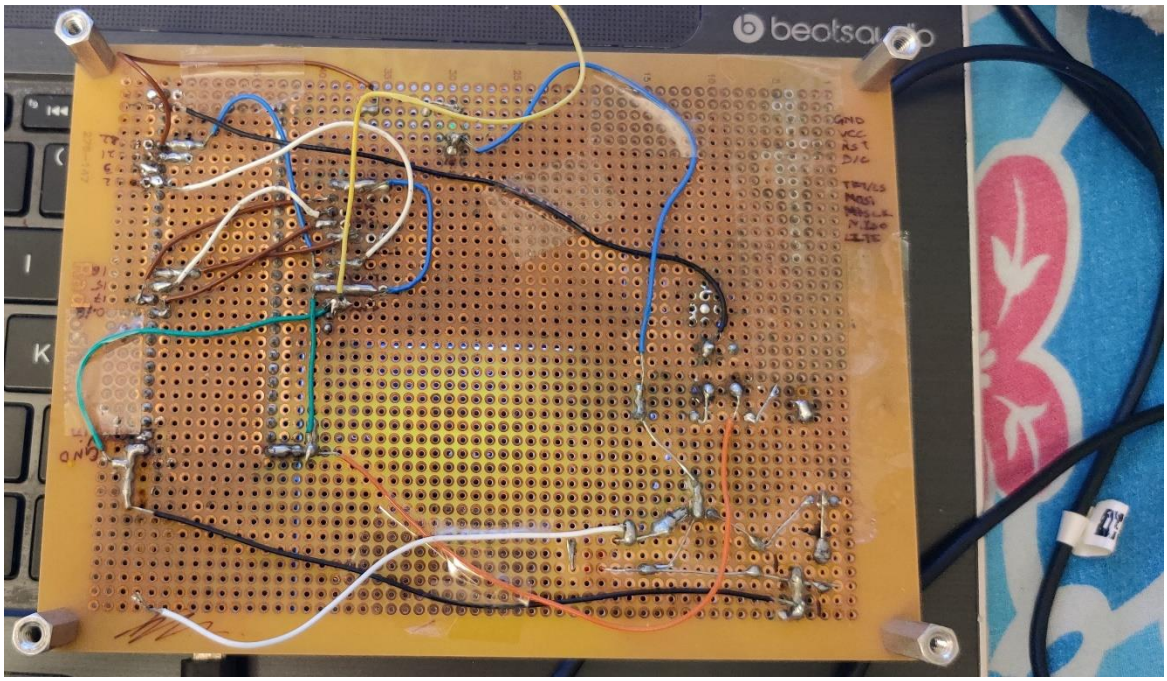

Fig. 5: Prototype Board Back

3. Figure 6 shows the laptop with code and the prototype board containing LPC1769 and LCD display.

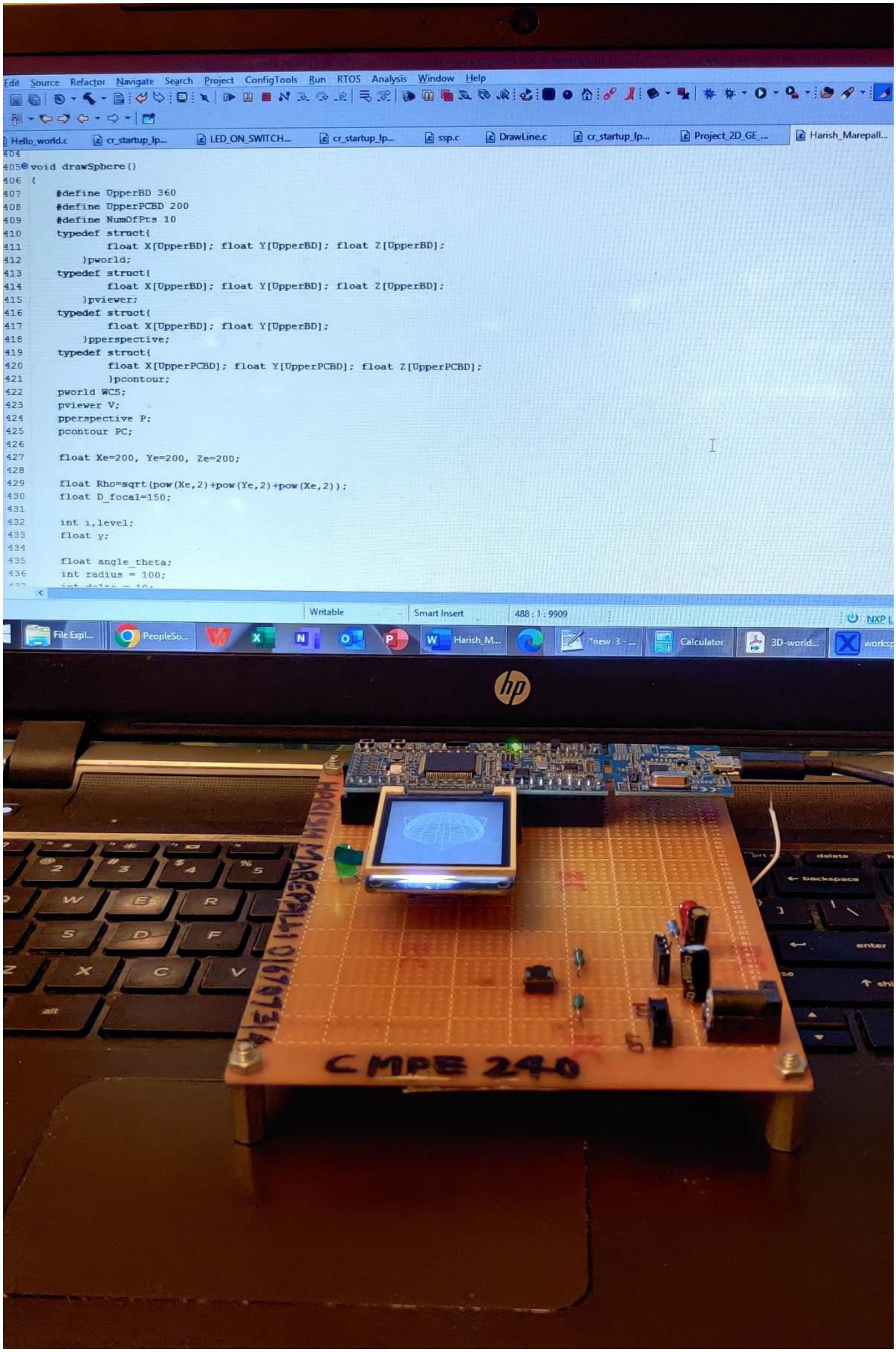

Fig. 6: Laptop with code and Prototype Board

4. Below figures show the cube and sphere for different axes values and D focal values. x_w-y_w-z_w axis and a floating cube is drawn. x_w axis is red, y_w axis is green, and z_w axis is blue. The lines of the cube are in white color.
   a. Axis lengths = 50.
   b. E(x,y,z) = (200,200,200)
   c. D = 50
   d. Floating cube vertices = (100,100,110)
   e. Sphere: Circle radius = 100
   f. 10 countours
   g. z_w distance = Depends on the level

   Figure 7 shows the cube and sphere with the above values

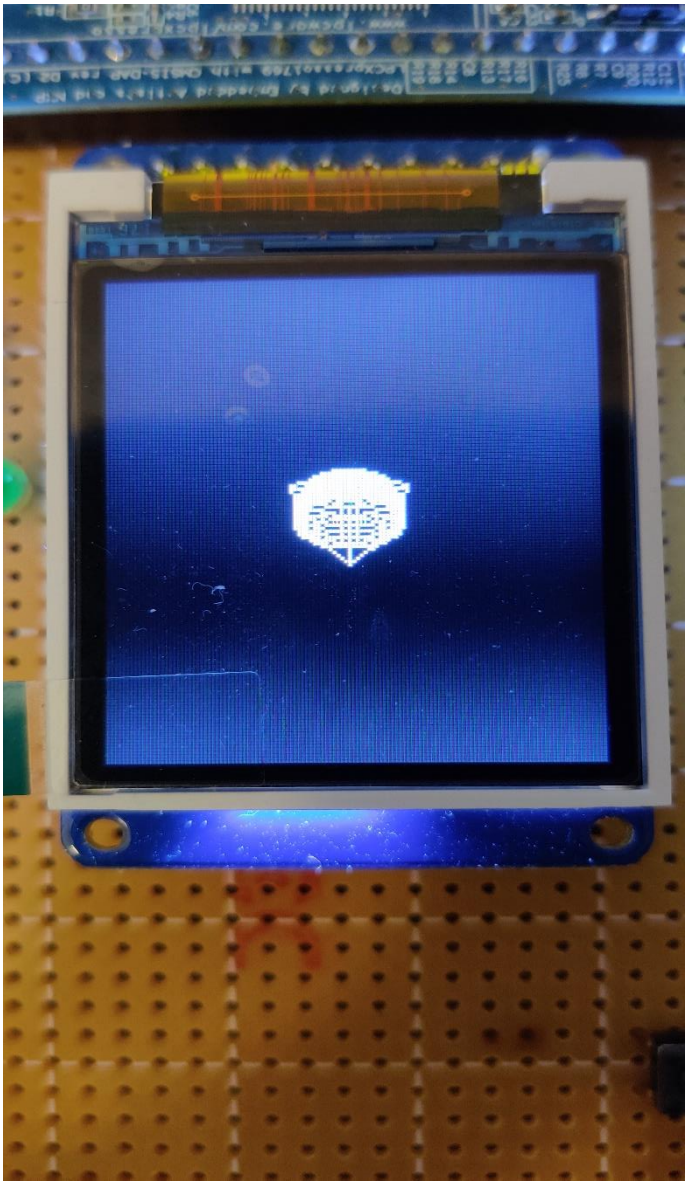

Fig. 7: Cube with Sphere (D=50, Axis=50)

5. Figure 8 shows the code, prototype board, cube and sphere with the above values.

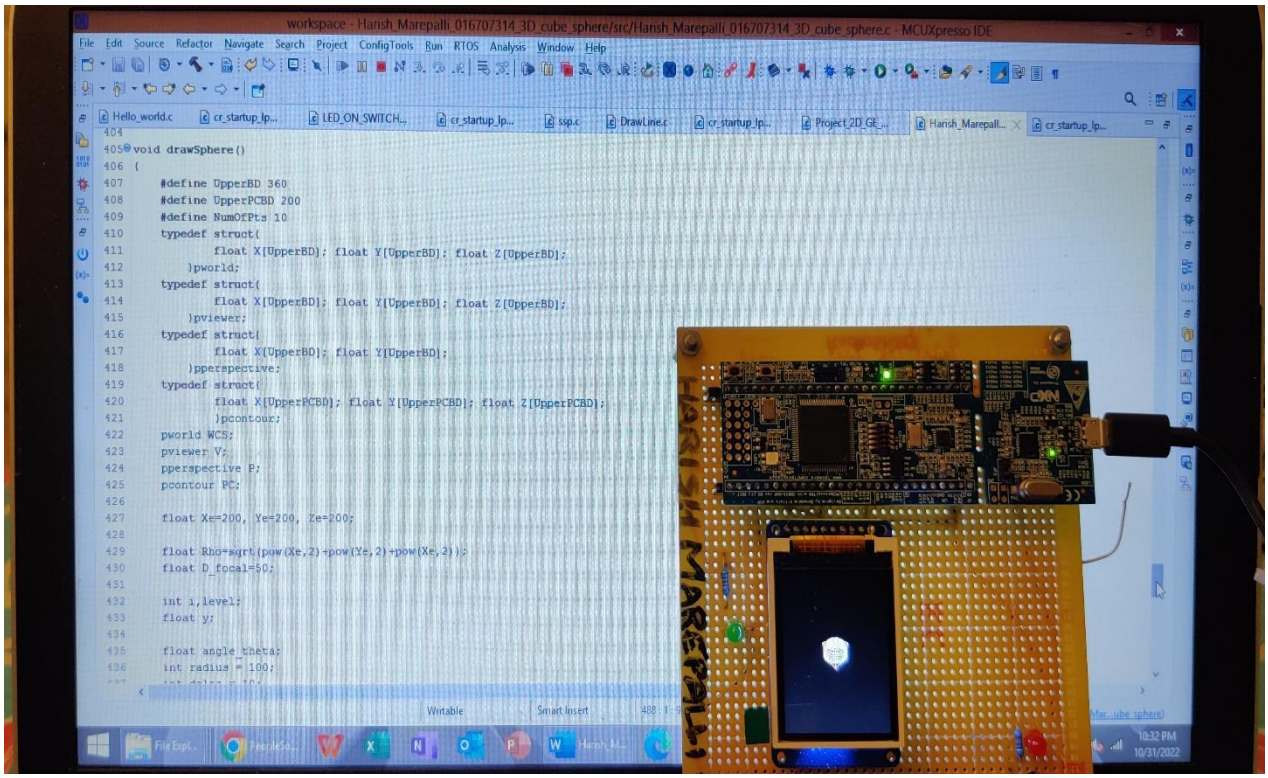

Fig. 8: Code, Prototype board, Cube, and Sphere (D=50, Axis=50)

6. Below figures show the cube and sphere for different axes values and D focal values. x_w-y_w-z_w axis and a floating cube is drawn. x_w axis is red, y_w axis is green, and z_w axis is blue. The lines of the cube are in white color.
   h. Axis lengths = 150.
   i. E(x,y,z) = (200,200,200)
   j. D = 150
   k. Floating cube vertices = (100,100,110)
   l. Sphere: Circle radius = 100
   m. 10 countours
   n. z_w distance = Depends on the level

Figure 9 shows the cube and sphere with the above values.

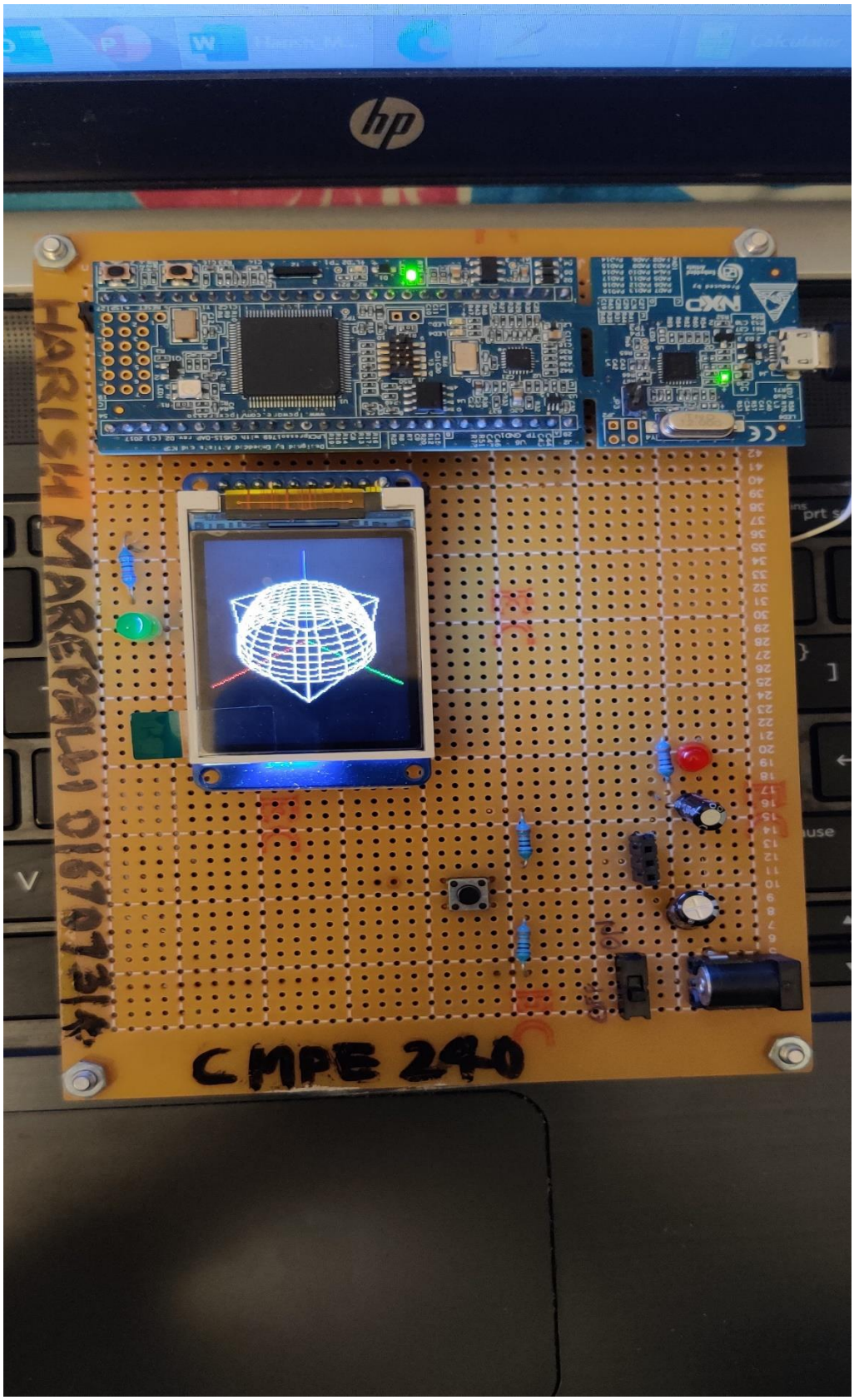

Fig. 9: Cube with Sphere (D=150, Axis=150)

7. Figure 10 shows the code, prototype board, cube and sphere with the above values.

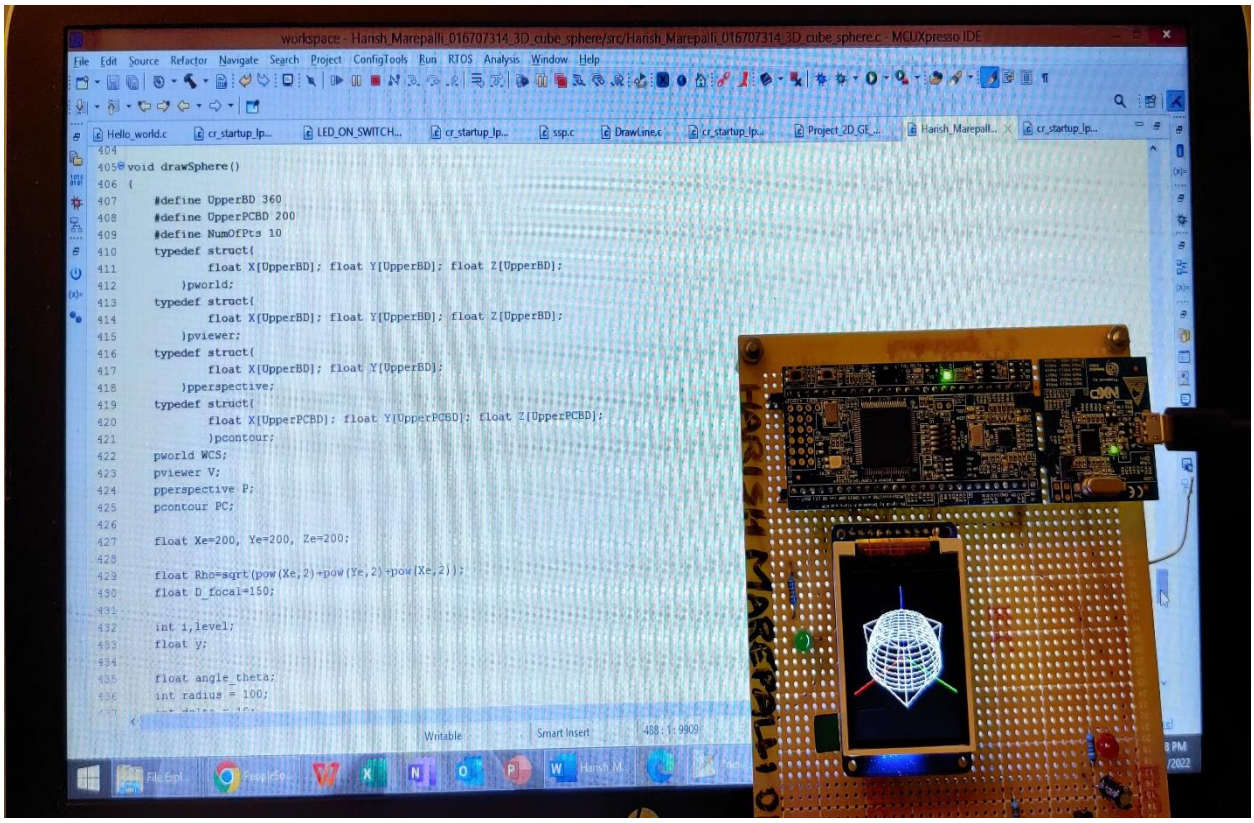

Fig. 10: Code, Prototype board, Cube, and Sphere (D=150, Axis=150)

8. Figure 11 shows the code snippet with only the editor.

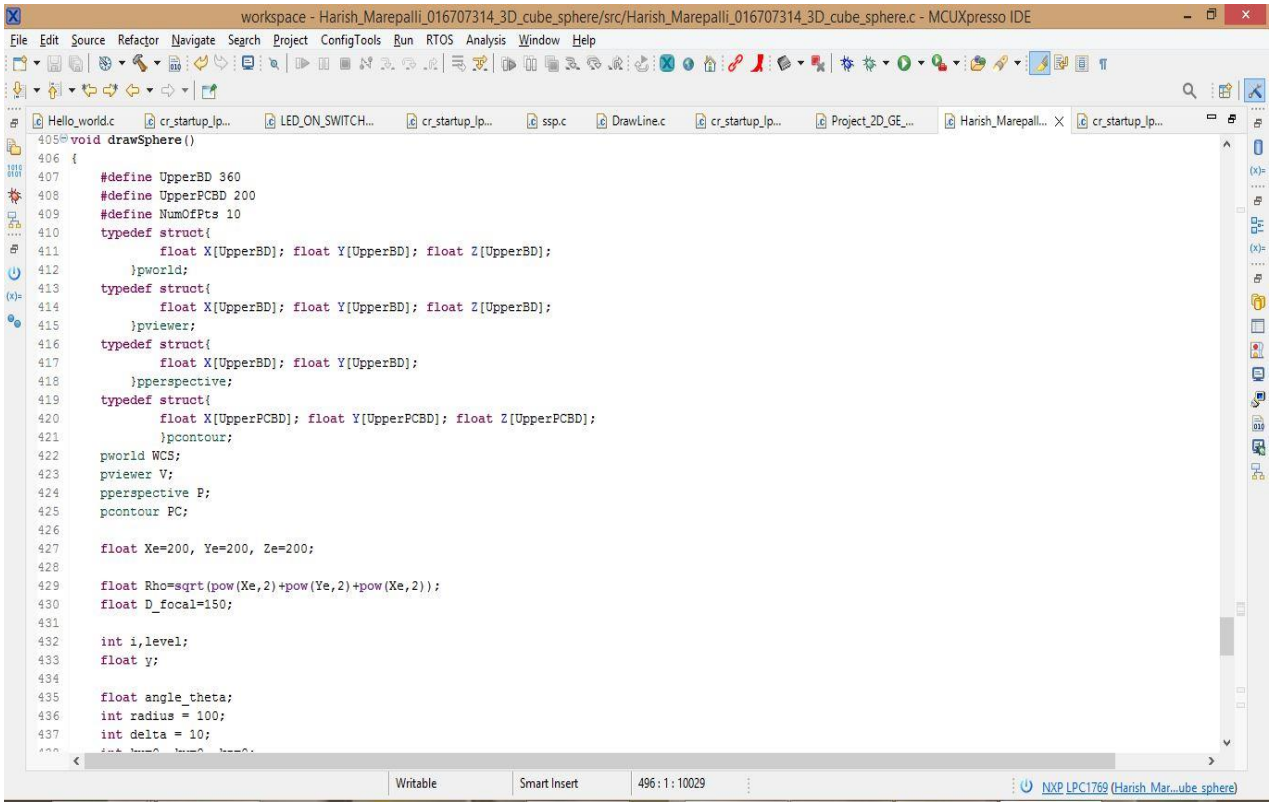

Fig. 11: Code Editor

9. Figure 12 shows the code snippet along with console.

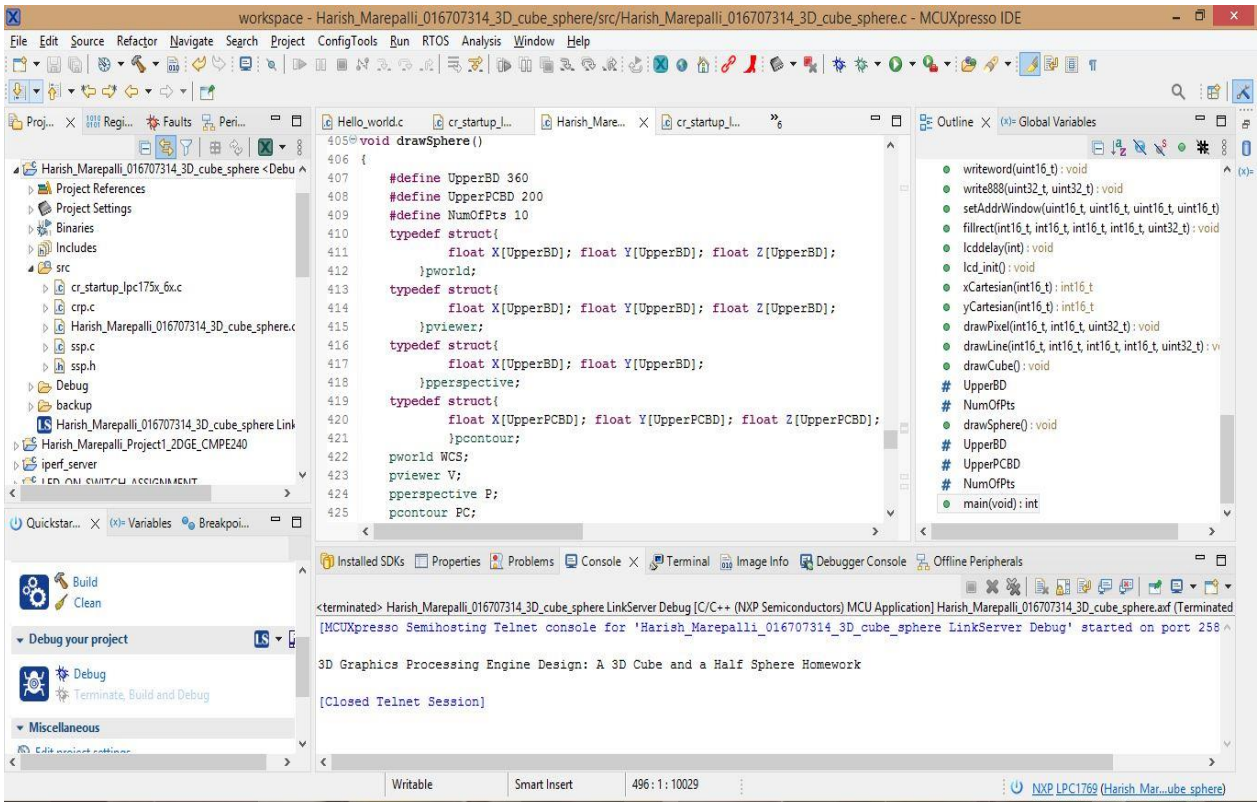

Fig. 12: Code with Console

CONCLUSION

This project is designed to make use of the LPC1769 CPU along with its inbuilt SPI controller to allow the implementation of 3D vector graphics on to LCD module. Upon completion of the experiment, I understood how to implement desired graphics using vector concepts and communicate it with LCD for the purpose of displaying. The project related source code is included in the appendix.

APPENDIX

SOURCE CODE

```
/*
===============================================================================
 Name        : Harish_Marepalli_016707314_3D_cube_sphere.c
 Author      : Harish Marepalli
 Version     :
 Copyright   : $(copyright)
 Description : This program is used to for creating and displaying the world
coordinate system, a 3D cube and a half sphere object based on the transformation
pipeline.
===============================================================================
*/

#include <cr_section_macros.h>
#include <NXP/crp.h>
#include "LPC17xx.h"                        /* LPC17xx definitions */
#include "ssp.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

/* Be careful with the port number and location number, because some of the
locations may not exist in that port. */

#define PORT_NUM            0


uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];


#define ST7735_TFTWIDTH 127
#define ST7735_TFTHEIGHT 159

#define ST7735_CASET 0x2A
#define ST7735_RASET 0x2B
#define ST7735_RAMWR 0x2C
#define ST7735_SLPOUT 0x11
#define ST7735_DISPON 0x29



#define swap(x, y) {x = x + y; y = x - y; x = x - y ;}

// Defining Color Values
#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF

// Custom Defined Colors
#define BROWN 0xA52A2A
#define YELLOW 0xFFFE00

// Custom Defined GREEN Shades
#define GREEN1 0x38761D
```

```
#define GREEN2 0x002200
#define GREEN3 0x00FC7C
#define GREEN4 0x32CD32
#define GREEN5 0x228B22
#define GREEN6 0x006400

// Custom Defined RED Shades
#define RED1 0xE61C1C
#define RED2 0xEE3B3B
#define RED3 0xEF4D4D
#define RED4 0xE88080

int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;

#define pi 3.1416

typedef struct Point
{
      float x;
      float y;
}Point;

void spiwrite(uint8_t c)
{
       int pnum = 0;

       src_addr[0] = c;

       SSP_SSELToggle( pnum, 0 );

       SSPSend( pnum, (uint8_t *)src_addr, 1 );

       SSP_SSELToggle( pnum, 1 );
}

void writecommand(uint8_t c)
{
       LPC_GPIO0->FIOCLR |= (0x1<<2);

       spiwrite(c);
}

void writedata(uint8_t c)
{
       LPC_GPIO0->FIOSET |= (0x1<<2);

       spiwrite(c);
}

void writeword(uint16_t c)
{
       uint8_t d;

       d = c >> 8;

       writedata(d);

       d = c & 0xFF;

       writedata(d);
}

void write888(uint32_t color, uint32_t repeat)
{
       uint8_t red, green, blue;
```

```
	int i;

	red = (color >> 16);

	green = (color >> 8) & 0xFF;

	blue = color & 0xFF;

	for (i = 0; i< repeat; i++)
	{
		writedata(red);

		writedata(green);

		writedata(blue);
	}
}

void setAddrWindow(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1)
{
	writecommand(ST7735_CASET);

	writeword(x0);

	writeword(x1);

	writecommand(ST7735_RASET);

	writeword(y0);

	writeword(y1);
}

void fillrect(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color)
{
	//int16_t i;

	int16_t width, height;

	width = x1-x0+1;

	height = y1-y0+1;

	setAddrWindow(x0,y0,x1,y1);

	writecommand(ST7735_RAMWR);

	write888(color,width*height);
}

void lcddelay(int ms)
{
	int count = 24000;

	int i;

	for ( i = count*ms; i > 0; i--);
}

void lcd_init()
{
	int i;
	printf("3D Graphics Processing Engine Design: A 3D Cube and a Half Sphere
Homework\n");
	// Set pins P0.16, P0.2, P0.22 as output
	LPC_GPIO0->FIODIR |= (0x1<<16);
```

```
        LPC_GPIO0->FIODIR |= (0x1<<2);

        LPC_GPIO0->FIODIR |= (0x1<<22);

        // Hardware Reset Sequence
        LPC_GPIO0->FIOSET |= (0x1<<22);
        lcddelay(500);

        LPC_GPIO0->FIOCLR |= (0x1<<22);
        lcddelay(500);

        LPC_GPIO0->FIOSET |= (0x1<<22);
        lcddelay(500);

        // initialize buffers
        for ( i = 0; i < SSP_BUFSIZE; i++ )
        {
                src_addr[i] = 0;
                dest_addr[i] = 0;
        }

        // Take LCD display out of sleep mode
        writecommand(ST7735_SLPOUT);
        lcddelay(200);

        // Turn LCD display on
        writecommand(ST7735_DISPON);
        lcddelay(200);
}

// Coordinate to Cartesian
int16_t xCartesian(int16_t x)
{
      x = x + (_width>>1);
      return x;
}

int16_t yCartesian(int16_t y)
{
      y = (_height>>1) - y;
      return y;
}

void drawPixel(int16_t x, int16_t y, uint32_t color)
{
        // Cartesian Implementation
        x=xCartesian(x); y=yCartesian(y);

        if ((x < 0) || (x >= _width) || (y < 0) || (y >= _height))

        return;

        setAddrWindow(x, y, x + 1, y + 1);

        writecommand(ST7735_RAMWR);

        write888(color, 1);
}

/*************************************************************************


** Descriptions:        Draw line function

**

** parameters:           Starting point (x0,y0), Ending point(x1,y1) and color
```

```
** Returned value:        None

**

*****************************************************************************/

void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color)
{
       int16_t slope = abs(y1 - y0) > abs(x1 - x0);

       if (slope)
       {
              swap(x0, y0);

              swap(x1, y1);
       }

       if (x0 > x1)
       {
              swap(x0, x1);

              swap(y0, y1);
       }

       int16_t dx, dy;

       dx = x1 - x0;

       dy = abs(y1 - y0);

       int16_t err = dx / 2;

       int16_t ystep;

       if (y0 < y1)
       {
             ystep = 1;
       }

       else
       {
             ystep = -1;
       }

       for (; x0 <= x1; x0++)
       {
              if (slope)
              {
                    drawPixel(y0, x0, color);
              }

              else
              {
                    drawPixel(x0, y0, color);
              }

              err -= dy;

              if (err < 0)
              {
                     y0 += ystep;

                     err += dx;
              }
       }
}
```

```
void drawCube()
{
	#define UpperBD 52
	#define NumOfPts 10
	typedef struct{
			float X[UpperBD]; float Y[UpperBD]; float Z[UpperBD];
		}pworld;
	typedef struct{
			float X[UpperBD]; float Y[UpperBD]; float Z[UpperBD];
		}pviewer;
	typedef struct{
			float X[UpperBD]; float Y[UpperBD];
		}pperspective;

	// EYE VIEW(200,200,200),

	float Xe=200, Ye=200, Ze=200;

	float Rho=sqrt(pow(Xe,2)+pow(Ye,2)+pow(Xe,2));
	float D_focal=150; //For perspective projection

	pworld WCS;
	pviewer V;
	pperspective P;

	// X-Y-Z World Coordinate System with each of the axis equal to 150

	// Origin
	WCS.X[0]=0.0; WCS.Y[0]=0.0; WCS.Z[0]=0.0;
	WCS.X[1]=150.0; WCS.Y[1]=0.0; WCS.Z[1]=0.0;
	WCS.X[2]=0.0; WCS.Y[2]=150.0; WCS.Z[2]=0.0;
	WCS.X[3]=0.0; WCS.Y[3]=0.0;   WCS.Z[3]=150.0;

	// Elevate Cube along Z_w axis by 10

	WCS.X[4]=100.0; WCS.Y[4]=0; WCS.Z[4]=10.0;
	WCS.X[5]=0.0; WCS.Y[5]=100.0; WCS.Z[5]=10.0;
	WCS.X[6]=0.0; WCS.Y[6]=0.0; WCS.Z[6]=110.0;
	WCS.X[7]=100.0; WCS.Y[7]=0.0; WCS.Z[7]=110.0;
	WCS.X[8]=100.0; WCS.Y[8]=100.0; WCS.Z[8]=10.0;
	WCS.X[9]=0.0; WCS.Y[9]=100.0; WCS.Z[9]=110.0;
	WCS.X[10]=100.0; WCS.Y[10]=100.0; WCS.Z[10]=110.0;

	float sPheta = Ye/sqrt(pow(Xe,2)+pow(Ye,2));
	float cPheta = Xe/sqrt(pow(Xe,2)+pow(Ye,2));
	float sPhi = sqrt(pow(Xe,2)+pow(Ye,2))/Rho;
	float cPhi = Ze/Rho;

	// WORLD TO VIEWER TRANSFROM

	for(int i=0;i<=NumOfPts;i++)
	{
		V.X[i] = -sPheta * WCS.X[i] + cPheta * WCS.Y[i];
		V.Y[i] = -cPheta * cPhi * WCS.X[i] - cPhi * sPheta * WCS.Y[i] + sPhi *
WCS.Z[i];
		V.Z[i] = -sPhi * cPheta * WCS.X[i] - sPhi * cPheta * WCS.Y[i] - cPhi *
WCS.Z[i] + Rho;
	}

	for(int i=0;i<=NumOfPts;i++)
	{
		P.X[i]=V.X[i]*(D_focal/V.Z[i]);
		P.Y[i]=V.Y[i]*(D_focal/V.Z[i]);
	}

	// CUBE DRAWLINES
```

```
	drawLine(P.X[0],P.Y[0],P.X[1],P.Y[1],RED);
	drawLine(P.X[0],P.Y[0],P.X[2],P.Y[2],0x00FF00);
	drawLine(P.X[0],P.Y[0],P.X[3],P.Y[3],0x0000FF);
	drawLine(P.X[7],P.Y[7],P.X[4],P.Y[4],WHITE);
	drawLine(P.X[7],P.Y[7],P.X[6],P.Y[6],WHITE);
	drawLine(P.X[7],P.Y[7],P.X[10],P.Y[10],WHITE);
	drawLine(P.X[8],P.Y[8],P.X[4],P.Y[4],WHITE);
	drawLine(P.X[8],P.Y[8],P.X[5],P.Y[5],WHITE);
	drawLine(P.X[8],P.Y[8],P.X[10],P.Y[10],WHITE);
	drawLine(P.X[9],P.Y[9],P.X[6],P.Y[6],WHITE);
	drawLine(P.X[9],P.Y[9],P.X[5],P.Y[5],WHITE);
	drawLine(P.X[9],P.Y[9],P.X[10],P.Y[10],WHITE);
}

void drawSphere()
{
	#define UpperBD 360
	#define UpperPCBD 200
	#define NumOfPts 10
	typedef struct{
			float X[UpperBD]; float Y[UpperBD]; float Z[UpperBD];
		}pworld;
	typedef struct{
			float X[UpperBD]; float Y[UpperBD]; float Z[UpperBD];
		}pviewer;
	typedef struct{
			float X[UpperBD]; float Y[UpperBD];
		}pperspective;
	typedef struct{
			float X[UpperPCBD]; float Y[UpperPCBD]; float Z[UpperPCBD];
			}pcontour;
	pworld WCS;
	pviewer V;
	pperspective P;
	pcontour PC;

	float Xe=200, Ye=200, Ze=200;

	float Rho=sqrt(pow(Xe,2)+pow(Ye,2)+pow(Xe,2));
	float D_focal=150;

	int i,level;
	float y;

	float angle_theta;
	int radius = 100;
	int delta = 10;
	int kx=0, ky=0, kz=0;

	for(level=0;level<=9;level++) //10 levels of cross section contours
	{
		for(i=0;i<360;i++)
		{
			angle_theta = i*3.142 /180;
			WCS.X[i] = 0 + radius*cos(angle_theta);
			WCS.Y[i] =  0 + radius*sin(angle_theta);
			WCS.Z[i] = 10*level; //Contours one above the other with the
distance of 10.
		}

		float sPheta = Ye/sqrt(pow(Xe,2)+pow(Ye,2));
		float cPheta = Xe/sqrt(pow(Xe,2)+pow(Ye,2));
		float sPhi = sqrt(pow(Xe,2)+pow(Ye,2))/Rho;
		float cPhi = Ze/Rho;

		// WORLD TO VIEWER TRANSFROM
```

```
            for(i=0;i<360;i++)
            {
                  V.X[i] = -sPheta * WCS.X[i] + cPheta * WCS.Y[i];
                  V.Y[i] = -cPheta * cPhi * WCS.X[i] - cPhi * sPheta * WCS.Y[i] +
sPhi * WCS.Z[i];
                  V.Z[i] = -sPhi * cPheta * WCS.X[i] - sPhi * cPheta * WCS.Y[i] -
cPhi * WCS.Z[i] + Rho;
            }

            for(i=0;i<360;i++)
            {
                  P.X[i]=V.X[i]*(D_focal/V.Z[i]);
                  P.Y[i]=V.Y[i]*(D_focal/V.Z[i]);
                  drawPixel(P.X[i], P.Y[i], WHITE);
                  if(i%18 == 0)
                  {
                        PC.X[kx++] = P.X[i];
                        PC.Y[ky++] = P.Y[i];
                  }
            }

            //radius-=10;
            radius -= (level + 1);
      }
      for(i=0;i<kx;i++)
      {
            if((i+20)<kx)
            drawLine(PC.X[i], PC.Y[i], PC.X[i+20], PC.Y[i+20], WHITE);
      }
}

int main (void)
{
      uint32_t pnum = 0 ;

      if ( pnum == 0 )
            SSP0Init();
      else
             puts("Port number is incorrect");

       lcd_init();

       fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, BLACK);

       drawCube();

       drawSphere();

       return 0;
}
```